Bootstrapping Skills

Daniel J. Mankowitz
Technion - Israel Institute of Technology
Electrical Engineering Department, The Technion - Israel Institute of Technology, Haifa 32000, Israel
danielm@tx.technion.ac.il

Timothy A. Mann
Google Deepmind
London, UK
timothymann@google.com

Shie Mannor
Technion - Israel Institute of Technology
Electrical Engineering Department, The Technion - Israel Institute of Technology, Haifa 32000, Israel
shie@ee.technion.ac.il

Abstract

The monolithic approach to policy representation in Markov Decision Processes (MDPs) looks for a single policy that can be represented as a function from states to actions. For the monolithic approach to succeed (and this is not always possible), a complex feature representation is often necessary since the policy is a complex object that has to prescribe what actions to take all over the state space. This is especially true in large domains with complicated dynamics. It is also computationally inefficient to both learn and plan in MDPs using a complex monolithic approach. We present a different approach where we restrict the policy space to policies that can be represented as combinations of simpler, parameterized skills—a type of temporally extended action, with a simple policy representation. We introduce Learning Skills via Bootstrapping (LSB) that can use a broad family of Reinforcement Learning (RL) algorithms as a "black box" to iteratively learn parametrized skills. Initially, the learned skills are short-sighted but each iteration of the algorithm allows the skills to bootstrap off one another, improving each skill in the process. We prove that this bootstrapping process returns a near-optimal policy. Furthermore, our experiments demonstrate that LSB can solve MDPs that, given the same representational power, could not be solved by a monolithic approach. Thus, planning with learned skills results in better policies without requiring complex policy representations.

1 Introduction

State-of-the-art Reinforcement Learning (RL) algorithms need to produce compact solutions to large or continuous state Markov Decision Processes (MDPs), where a solution, called a policy, generates an action when presented with the current state. One such approach to producing compact solutions is linear function approximation.

MDPs are important for both planning and learning in *Reinforcement Learning (RL)*. The RL planning problem uses an MDP model to derive a policy that maximizes the sum of rewards received, while the RL learning problem learns an MDP model from experience (because the MDP model is unknown in advance). In this paper, we focus on RL planning, and use insights from RL that could be used to scale up to problems that are unsolvable with traditional planning approaches (such as Value Iteration and Policy Iteration (c.f., Puterman [1994]). A general result from ma-

chine learning is that the sample complexity of learning increases with the complexity of the representation Vapnik [1998]. In a planning scenario, increased sample complexity directly translates to an increase in computational complexity. Thus monolithic approaches, which learn a single parametric policy that solves the entire MDP, scale poorly. This is because they often require highly complex feature representations, especially in high-dimensional domains with complicated dynamics, to support near-optimal policies. Instead, we investigate learning a collection of policies over a much simpler feature representation (compact policies) and combine those policies hierarchically.

Generalization: the ability of a system to perform accurately on unseen data, is important for machine learning in general, and can be achieved in this context by restricting the policy space, resulting in compact policies Bertsekas [1995], Sutton [1996]. Policy Search (PS) algorithms, a form of generalization, learn and maintain a compact policy representation so that the policy generates similar actions in nearby states Peters and Schaal [2008], Bhatnagar et al. [2009].

Temporally Extended Actions [TEAs, Sutton et al., 1999]: Compact policies can be represented and combined hierarchically as TEAs. TEAs are control structures that execute for multiple timesteps. They have been extensively studied under different names, including skills Konidaris and Barto [2009], macro-actions Hauskrecht et al. [1998], He et al. [2011], and options Sutton et al. [1999]. TEAs are known to speed up the convergence rate of MDP planning algorithms Sutton et al. [1999], Mann and Mannor [2014]. However, the effectiveness of planning with TEAs depends critically on the given actions. For example, Figure 1a depicts an episodic MDP with a single goal region and skills $\{\sigma_1, \sigma_2, \ldots, \sigma_5\}$. In this 2D setting, each skill represents a simple movement in a single, linear direction. Most of the TEAs move towards the goal region, but σ_5 moves in the opposite direction of the goal making it impossible to reach. With these TEAs, we cannot hope to derive a satisfactory solution. On the other hand, if one of the TEAs takes the agent directly to the goal (Figure 1b, the monolithic approach), then planning becomes trivial. Notice, however, that this TEA may be quite complex, and therefore difficult to learn since, in this 2D setting, it represents non-linear movements in multiple directions.

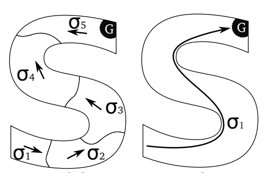


Figure 1: TEAs in an episodic MDP with S-shaped state-space and goal region G. (a) Although most actions move toward the goal, σ_5 moves away from the goal making it impossible to complete the task. (b) Planning becomes trivial when a single *TEA* takes the agent directly to G.

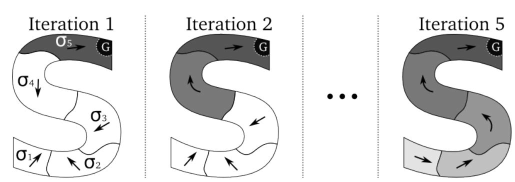


Figure 2: A partitioning of a target MDP in the pinball domain. Each sub-partition (partition class) i represents the skill MDP M'_i.

Learning a useful set of TEAs has been a topic of intense research McGovern and Barto [2001], Moerman [2009], Konidaris and Barto [2009], Brunskill and Li [2014], Hauskrecht et al. [1998]. However, prior work suffers from the following drawbacks: (1) lack of theoretical analysis guaranteeing that the derived policy will be near-optimal in continuous state MDPs, (2) the process of learning TEAs is so expensive that it needs to be ammortized over a sequence of MDPs, (3) the approach is not applicable to MDPs with large or continuous state-spaces, or (4) the learned TEAs do not generalize over the state-space. We provide the first theoretical guarantees for iteratively learning a set of simple, generalizable parametric TEAs (skills) in a continuous state MDP. The learned TEAs solve the given tasks in a near-optimal manner.

Skills: Generalization & Temporal Abstraction: Skills are TEAs defined over a parametrized policy. Thus, they incorporate both temporal abstraction and generalization. As TEAs, skills are closely related to options Sutton et al. [1999] developed in the RL literature. In fact, skills, as defined here, are a special case of options. Therefore, skills inherit many of the useful theoretical properties of options (e.g., Precup et al. [1998]). The main difference between skills and more general options is that skills are based on parametric policies that can be initialized and reused in any region of the state space.

We introduce a novel meta-algorithm, Learning Skills via Bootstrapping (LSB), that uses an RL algorithm as a "black box" to iteratively learn parametrized skills. The learning algorithm is given a partition of the state-space, and one skill is created for each class in the partition. This is a very weak requirement since any partition could be used, such as a grid. During an iteration, an RL algorithm is used to update each skill. The skills may be initialized arbitrarily, but

after the first iteration skills with access to a goal region or non-zero rewards will learn how to exploit those rewards (e.g., Figure 2, Iteration 1). On further iterations, the newly acquired skills propagate reward back to other regions of the state-space. Thus, skills that previously had no reward signal bootstrap off of the rewards of other skills (e.g., Figure 2, Iterations 2 and 5). Although each skill is only learned over a single partition class, it can be initialized in any state.

It is important to note that this paper deals primarily with learning *TEAs* or *Skills* that aid in both speeding up the convergence rate of RL planning algorithms Sutton et al. [1999], Mann and Mannor [2014], as well as enabling larger problems to be solved using skills with simple policy representations. Utilizing simple policy representations is advantageous since this results in better generalization and better sample efficiency. These *skills* represent a misspecified model of the problem since they are not known in advance. By learning skills, we are also therefore inherently tackling the learning problem as we are iteratively correcting a misspecified model.

Contributions: Our main contributions are **(1)** The introduction of Learning Skills via Bootstrapping (LSB), which requires no additional prior knowledge apart from a partition over the state-space. **(2)** LSB is the first algorithm for learning skills in continuous state-spaces with theoretical convergence guarantees. **(3)** Theorem 1, which relates the quality of the policy returned by LSB to the quality of the skills learned by the "black box" RL algorithm. **(4)** Experiments demonstrating that LSB can solve MDPs that, given the same representational power, can not be solved by a policy derived from a monolithic approach. Thus, planning with learned skills allows us to work with simpler representations Barto et al. [2013], which ultimately allows us to solve larger MDPs.

2 Background

Let $M = \langle S, A, P, R, \gamma \rangle$ be an MDP, where S is a (possibly infinite) set of states, A is a finite set of actions, P is a mapping from state-action pairs to probability distributions over next states, R maps each state-action pair to a reward in $[0, 1]$, and $\gamma \in [0, 1)$ is the discount factor. While assuming the rewards are in $[0, 1]$ may seem restrictive, any bounded space can be rescaled so that this assumption holds. A policy $\pi(a|s)$ gives the probability of executing action $a \in A$ from state $s \in S$.

Let M be an MDP. The value function of a policy π with respect to a state $s \in S$ is $V^{\pi}_M(s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} R(s_t, a_t) | s_0 = s\right]$ where the expectation is taken with respect to the trajectory produced by following policy π. The value function of a policy π can also be written recursively as

$$V^{\pi}_M(s) = \mathbb{E}_{a \sim \pi(\cdot|s)}\left[R(s,a)\right] + \gamma \mathbb{E}_{s' \sim P(\cdot|s,\pi)}\left[V^{\pi}(s')\right] \quad , \tag{1}$$

which is known as the Bellman equation. The optimal Bellman equation can be written as $V^*_M(s) = \max_a \mathbb{E}\left[R(s,a)\right] + \gamma \mathbb{E}_{s' \sim P(\cdot|s,\pi)}\left[V^*(s')\right]$. Let $\varepsilon > 0$. We say that a policy π is ε-optimal if $V^{\pi}_M(s) \geq V^*_M(s) - \varepsilon$ for all $s \in S$. The action-value function of a policy π can be defined by $Q^{\pi}_M(s,a) = \mathbb{E}_{a \sim \pi(\cdot|s)}\left[R(s,a)\right] + \gamma \mathbb{E}_{s' \sim P(\cdot|s,\pi)}\left[V^{\pi}(s')\right]$, for a state $s \in S$ and an action $a \in A$, and the optimal action-value function is denoted by $Q^*_M(s,a)$. Throughout this paper, we will drop the dependence on M when it is clear from context.

3 Skills

One of the key ideas behind skills is that they may be learned locally, but they can be used throughout the entire state-space. We present a new formal definition for skills and a skill policy.

Definition 1. *A* **skill** *σ is defined by a pair $\langle \pi_\theta, \beta \rangle$, where π_θ is a parametric policy with parameter vector θ and $\beta : S \to \{0, 1\}$ indicates whether the skill has finished (i.e., $\beta(s) = 1$) or not (i.e., $\beta(s) = 0$) given the current state $s \in S$.*

Definition 2. *Let Σ be a set of $m \geq 1$ skills. A* **skill policy** *μ is a mapping $\mu : S \to [m]$ where S is the state-space and $[m]$ is the index set over skills.*

A skill policy selects which skill to initialize from the current state by returning the index of one of the skills. By defining skill policies to select an index (rather than the skill itself), we can use the same policy even as the set of skills is adapting. Next we define a *Skill MDP*, which is a sub-partition of a target MDP as shown in Figure 3.

Definition 3. *Given a target MDP $M = \langle S, A, P, R, \gamma \rangle$ and value function V_M, a* **Skill MDP** *for partition $\mathcal{P}_i$ is an MDP defined by $M'_i = \langle S', A, P', R', \gamma \rangle$ where $S' = \mathcal{P}_i \cup \{s_T\}$ where s_T is a terminal state and A is the action set from M. The transition probability function $P'(s'|s,a)$ and reward function $R'(s,a)$ are defined below.*

$$P'(s'|s,a) = \begin{cases} P(s'|s,a) & \text{if } s \in \mathcal{P}_i \wedge s' \in \mathcal{P}_i \\ \sum_{y \in S \setminus \mathcal{P}_i} P(y|s,a) & \text{if } s \in \mathcal{P}_i \wedge s' = s_T \\ 1 & \text{if } s = s_T \wedge s' = s_T \\ 0 & \text{if } s = s_T \wedge s' \neq s_T \end{cases}, \quad R'(s,a) = \begin{cases} 0 & \text{if } s = s_T \\ \sum\limits_{s' \in \mathcal{P}_i} P(s'|s,a) R(s,a) & \text{if } s \neq s_T \wedge s' \neq s_T \\ \sum\limits_{y \in S \setminus \mathcal{P}_i} \psi(s,a,y) & \text{if } s \neq s_T \wedge s' = s_T \end{cases},$$

where $\psi(s,a,y) = P(y|s,a)\left(R(s,a) + \gamma V_M(y)\right)$, and γ is the discount factor from M.

A Skill MDP M'_i is an episodic MDP that terminates once the agent escapes from $\mathcal{P}_i$ and upon terminating receives a reward equal to the value of the state the agent would have transitioned to in the target MDP. Therefore, we construct a modified MDP called a Skill MDP and apply a planning or RL algorithm to solve it. The resulting solution is a skill. Each Skill MDP M'_i is defined within the partition $\mathcal{P}_i$.

Given a good set of skills, planning can be significantly faster Sutton et al. [1999], Mann and Mannor [2014]. However, in many domains we may not be given a good set of skills. Therefore it is necessary to learn this set of skills given the unsatisfactory skill set. In the next section, we introduce an algorithm for dynamically improving skills via bootstrapping.

4 Learning Skills via Bootstrapping (LSB) Algorithm

Algorithm 1: **Learning Skills via Bootstrapping (LSB)**

Require: M {Target MDP}, $\mathcal{P}$ {Partitioning of S}, K {# Iterations}
1: $m \leftarrow |\mathcal{P}|$ {# of partitions.}
2: $\mu(s) = \arg\max_{i \in [m]} \mathbb{I}\{s \in \mathcal{P}_i\}$
3: Initialize Σ with m skills. {1 skill per partition.}
4: **for** $k = 1, 2, \ldots, K$ **do** {Do K iterations.}
5:   **for** $i = 1, 2, \ldots, m$ **do** {One update per skill.}
6:     **Policy Evaluation:**
7:     Evaluate μ with Σ to obtain $V_M^{\langle \mu, \Sigma \rangle}$
8:     **Skill Update:**
9:     Construct Skill MDP M'_i from M & $V_M^{\langle \mu, \Sigma \rangle}$
10:     Solve M'_i obtaining policy π_θ
11:     $\sigma'_i \leftarrow \langle \pi_\theta, \beta_i \rangle$
12:     Replace σ_i in Σ by σ'_i
13:   **end for**
14: **end for**
15: **return** $\langle \mu, \Sigma \rangle$

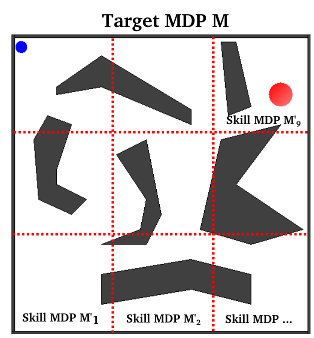


Figure 3: A partitioning of a target MDP in the pinball domain. Each sub-partition (partition class) i represents the skill MDP M'_i. Note that, so long as the classes overlap one another and the goal region is within one of the classes, near-optimal convergence is guaranteed. Therefore, the entire state-space does not have to be partitioned.

Learning Skills via Bootstrapping (LSB, Algorithm 1) takes a target MDP M, a partition $\mathcal{P}$ over the state-space and a number of iterations $K \geq 1$ and returns a pair $\langle \mu, \Sigma \rangle$ containing a skill policy μ and a set of skills Σ. The number of skills $m = |\mathcal{P}|$ is equal to the number of classes in the partition $\mathcal{P}$ (line 1). The skill policy μ returned by LSB is defined (line 2) by

$$\mu(s) = \arg\max_{i \in [m]} \mathbb{I}\left\{s \in \mathcal{P}_i\right\} \quad , \tag{2}$$

where $\mathbb{I}\{\cdot\}$ is the indicator function returning 1 if its argument is true and 0 otherwise and $\mathcal{P}_i$ denotes the i^{th} class in the partition $\mathcal{P}$. Thus μ simply returns the index of the skill associated with the partition class containing the current state. On line 3, LSB could either initialize Σ with skills that we believe might be useful or initialize them arbitrarily, depending on our level of prior knowledge.

Next (lines 4 – 14), LSB performs K iterations. In each iteration, LSB updates the skills in Σ (lines 5 – 13). Remember that the value of a skill depends on how it is combined with other skills (e.g., Figure 1a failed because a single TEA prevented reaching the goal). If we allowed all skills to change simultaneously, the skills could not reliably bootstrap off of each other. Therefore, LSB updates each skill individually. Multiple iterations are needed so that the skill set can converge (Figure 2).

The process of updating a skill (lines 6 – 12) starts by evaluating μ with the current skill set Σ (line 6). Any number of policy evaluation algorithms could be used here, such as TD(λ) with function approximation Sutton and Barto [1998] or LSTD Boyan [2002], modified to be used with skills. In our experiments, we used a straighforward variant of LSTD Sorg and Singh [2010]. Then we use the target MDP M to construct a Skill MDP M' (line 9). Next, LSB uses a planning or RL algorithm to approximately solve the Skill MDP M' returning a parametrized policy π_θ (line 10). Any planning or RL algorithm for regular MDPs could fill this role provided that it produces a parametrized policy. However, in our experiments, we used a simple actor-critic PG algorithm, unless otherwise stated. Then a new skill $\sigma'_i = \langle \pi_\theta, \beta_i \rangle$ is created (line 11) where π_θ is the policy derived on line 10 and $\beta_i(s) = \begin{cases} 0 & \text{if } s \in \mathcal{P}_i \\ 1 & \text{otherwise} \end{cases}$. The definition of β_i means that the skill will terminate only if it leaves the i^{th} partition. Finally, we update the skill set Σ by replacing the i^{th} skill with σ'_i (line 12). It is important to note that in LSB, updating a skill is equivalent to solving a Skill MDP.

5 Analysis of LSB

We provide the first convergence guarantee for iteratively learning skills in a continuous state MDP using LSB (Lemma 1 and Lemma 2, proven in the supplementary material). We use this guarantee as well as Lemma 2 to prove Theorem 1. This theorem enables us to analyze the quality of the policy returned by LSB. It turns out that the quality of the policy depends critically on the quality of the skill learning algorithm. An important parameter for determining the quality of a policy returned by LSB is the skill learning error defined below.

Definition 4. *Let $\mathcal{P}$ be a partition over the target MDP's state-space. The* **skill learning error** *is*

$$\eta_{\mathcal{P}} = \max_{i \in [m]} \eta_i \ , \tag{3}$$

where η_i is the smallest $\eta_i \geq 0$, such that
$V^*_{M'_i}(s) - V^{\pi_\theta}_{M'_i}(s) \leq \eta_i$, for all $s \in \mathcal{P}_i$ and π_θ is the policy returned by the skill learning algorithm executed on M'_i.

The skill learning error quantifies the quality of the Skill MDP solutions returned by our skill learning algorithm. If we used an exact solver to learn skills, then $\eta_{\mathcal{P}} = 0$. However, if we use an approximate solver, then $\eta_{\mathcal{P}}$ will be non-zero and the quality will depend on the partition $\mathcal{P}$. Generally, using finer grain partitions will decrease $\eta_{\mathcal{P}}$. However, Theorem 1 reveals that adding too many skills can also negatively impact the returned policy's quality.

Theorem 1. *Let $\varepsilon > 0$. If we run LSB with partition $\mathcal{P}$ for $K \geq \log_\gamma(\varepsilon(1-\gamma))$ iterations, then the algorithm returns policy $\varphi = \langle \mu, \Sigma \rangle$ such that*

$$\|V^*_M - V^\varphi_M\|_\infty \leq \frac{m\eta_{\mathcal{P}}}{(1-\gamma)^2} + \varepsilon \ , \tag{4}$$

where m is the number of classes in $\mathcal{P}$.

The proof of Theorem 1 is divided into three parts (a complete proof is given in the supplementary material). The main challenge to proving Theorem 1 is that updating one skill can have a significant impact on the value of other skills. Our analysis starts by bounding the impact of updating one skill. Note that Σ represents a skill set and Σ_i represents a skill set where we have updated the i^{th} skill (corresponding to the i^{th} partition class $\mathcal{P}_i$) in the set. (1) First, we show that error between V^*_M, the globally optimal value function, and $V^{\langle \mu, \Sigma_i \rangle}_M$, is a contraction when $s \in \mathcal{P}_i$ and is bound by $\|V^*_M - V^{\langle \mu, \Sigma \rangle}_M\|_\infty + \frac{\eta_{\mathcal{P}}}{1-\gamma}$ otherwise (Lemma 1). (2) Next we apply an inductive argument to show that updating all m skills results in a γ contraction over the entire state space (Lemma 2). (3) Finally, we apply this contraction recursively, which proves Theorem 1.

This provides the first theoretical guarantees of convergence to a near optimal solution when iteratively learning a set of skills Σ in a continuous state space. Theorem 1 tells us that when the skill learning error is small, LSB returns a near-optimal policy. The first term on the right hand side of (4) is the approximation error. This is the loss we pay for

the parametrized class of policies that we learn skills over. Since m represents the number of classes defined by the partition, we now have a formal way of analysing the effect of the partitioning structure. In addition, complex skills do not need to be designed by a domain expert; only the partitioning needs to be provided *a-priori*. The second term is the convergence error. It goes to 0 as the number of iterations K increases.

At first, the guarantee provided by Theorem 1 may appear similar to (Hauskrecht et al. [1998], Theorem 1). However, Hauskrecht et al. [1998] derive TEAs only at the beginning of the learning process and do not update them. On the other hand, LSB updates its skill set dynamically via bootstrapping. Thus, LSB does not require prior knowledge of the optimal value function.

Theorem 1 does not explicitly present the effect of policy evaluation error, which occurs with any approximate policy evaluation technique. However, if the policy evaluation error is bounded by $\nu > 0$, then we can simply replace $\eta_{\mathcal{P}}$ in (4) with $(\eta_{\mathcal{P}} + \nu)$. Again, smaller policy evaluation error leads to smaller approximation error.

6 Experiments and Results

We performed experiments on three well-known RL benchmarks: Mountain Car (MC), Puddle World (PW) Sutton [1996] and the Pinball domain Konidaris and Barto [2009]. The MC domain has similar results to PW and therefore has been moved to the supplementary material. We use two variations for the Pinball domain, namely *maze-world*, which we created, and *pinball-world* which is one of the standard pinball benchmark domains. Our experiments show that, using a simple policy representation, the monolithic approach is unable to adequately solve the tasks in each case as the policy representation is not complex enough. However, LSB can solve these tasks with the same simple policy representation by combining bootstrapped skills. These domains are simple enough that we can still solve them using richer representations. This allows us to compare LSB to a policy that is very close to optimal. Our experiments demonstrate potential to scale up to higher dimensional domains by combining skills over simple representations.

Recall that LSB is a meta-algorithm. We must provide an algorithm for Policy Evaluation (PE) and skill learning. In our experiments, for the MC and PW domains, we used SMDP-LSTD Sorg and Singh [2010] for PE and a modified version of Regular-Gradient Actor-Critic Bhatnagar et al. [2009] for skill learning (see supplementary material for details). In the Pinball domains, we used Nearest-Neighbor Function Approximation (NN-FA) for PE and UCB Random Policy Search (UCB-RPS) for skill learning.

In our experiments, for the MC and PW domains, each skill is simply represented as a probability distribution over actions (independent of the state). We compare their performance to a policy using the same representation that has been derived using the monolithic approach. Each experiment is run for 10 independent trials. A 2×2 grid partitioning is used for the skill partition in these domains, unless otherwise stated. Binary-grid features are used to estimate the value function. In the pinball domains, each skill is represented by 5 polynomial features corresponding to each state dimension and a bias term. A $4 \times 1 \times 1 \times 1$ grid-partitioning is used for *maze-world* and a $4 \times 3 \times 1 \times 1$ partitioning is used for *pinball-world*. The value function is represented by a KD-Tree containing 1000 state-value pairs uniformly sampled in the domain. A value for a particular state is obtained by assigning the value of the nearest neighbor to that state that is contained within the KD-tree. Each experiment in the pinball domain has been run for 5 independent trials. These are example representations. In principal, any value function and policy representation that is representative of the domain can be utilized.

6.1 Puddle World

Puddle World is a 2-dimensional world containing two puddles. A successful agent should navigate to the goal location, avoiding the puddles. The state space is the $\langle x, y \rangle$ location of the agent. Figure $4a$ compares the monolithic approach with LSB (for a 2×2 grid partition). The monolithic approach achieves low average reward. However, with the same restricted policy representation, LSB combines a set of skills, resulting in a richer solution space and a higher average reward as seen in Figure $4a$. This is comparable to the approximately optimal average reward attained by executing Approximate Value Iteration (AVI) for a huge number of iterations. In this experiment LSB is not initiated in the partition class containing the goal state but still achieves near-optimal convergence after only 2 iterations.

Figure $4b$ compares the performance of different partitions where a 1×1 grid represents the monolithic approach. The skill learning error η_P is significantly smaller for all the partitions greater than 1×1, resulting in lower cost. On the other hand, according to Theorem 1, adding more skills m increases the cost. A tradeoff therefore exists between η_P and m. In practice, η_P tends to dominate m. In addition to the tradeoff, the importance of the partition design is

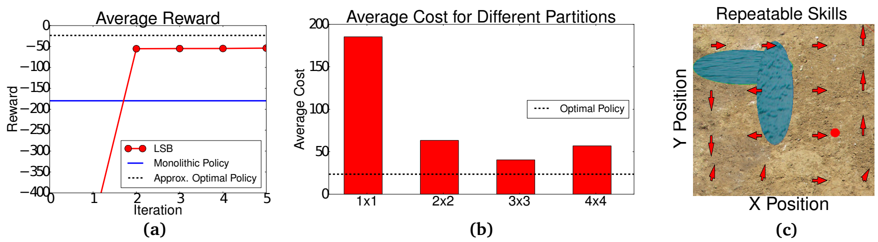


Figure 4: The Puddle World domain: (a) The average reward for the LSB algorithm generated by the LSB skill policy. This is compared to the monolithic approach that attempts to solve the global task as well as an approximately optimal policy derived using Q-learning (applied for a huge number of iterations). (b) The average cost (negative reward) for each grid partition. (c) Repeatable skills plot.

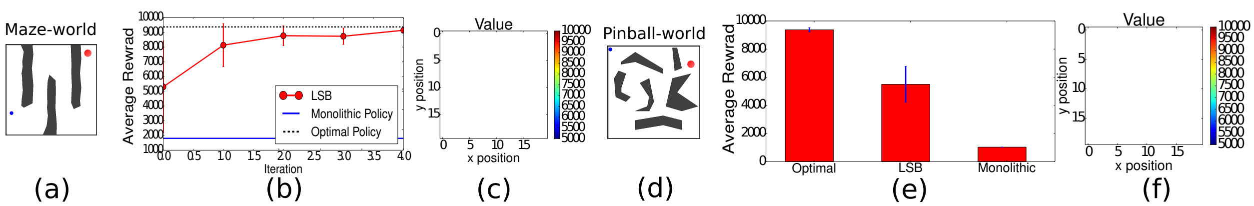


Figure 5: The Pinball domains: (a) The maze-world domain. (b) The average reward for the LSB algorithm generated by the LSB skill policy for *maze-world*. This is compared to the monolithic approach that attempts to solve the global task as well as an approximately optimal policy derived using Approximate Value Iteration (AVI) executed for a huge number of iterations. (c) The learned value function for the maze-world domain. (d) The pinball-world domain. (e) The average reward for the LSB algorithm generated by the LSB skill policy in the *pinball-world* domain. In this domain, LSB converges after a single iteration as we start LSB in the partition containing the goal. (f) The learned value function.

evident when analyzing the cost of the 3×3 and 4×4 grids. In this scenario, the 3×3 partition design is better suited to Puddle World than the 4×4 partition, resulting in lower cost.

6.2 Skill Generalization

In the worst case, the number of skills learned by (LSB) is based on the partition. However, LSB may learn similar skills in different partition classes, adding redundancy to the skill set. This suggests that skills can be reused in different parts of the state-space, resulting in less skills compared to the number of partition classes. To validate this intuition, a 4×4 grid was created for both the Mountain Car and Puddle World domains. We ran LSB using this grid on each domain. Since it is more intuitive to visualize and analyze the reusable skills generated for the 2D Puddle World, we present these skills in a quiver plot superimposed on the Puddle World (Figure $4c$). For each skill, the direction (red arrows in Figure $4c$) is determined by sampling and averaging actions from the skill's probability distribution. As can be seen in Figure $4c$, many of the learned skills have the same direction. These skills can therefore be combined into a single skill and reused throughout the state-space. In this case, the skill-set consisting of 16 skills can be reduced to a reusable skill-set of 5 skills (the four cardinal directions, including two skills that are in the approximately north direction). Therefore, skill reuse may further reduce the complexity of a solution.

6.3 Pinball

These experiments have been performed in domains with simple dynamics. We decided to test LSB on a domain with significantly more complicated dynamics, namely Pinball Konidaris and Barto [2009]. The goal in Pinball is to direct an agent (the blue ball) to the goal location (the red region). The Pinball domain provides a sterner test for LSB as the velocity at which the agent is travelling needs to be taken into account to circumnavigate obstacles. In addition, collisions with obstacles in the environment are non-linear at obstacle vertices. The state space is the four-tuple $\langle x, y, \dot{x}, \dot{y} \rangle$ where x, y represents the 2D location of the agent, and $\dot{x}, \dot{y}$ represents the velocities in each direction.

Two domains have been utilized, namely *maze-world* and *pinball-world* (Figure $5a$ and Figure $5d$ respectively). For *maze-world*, a $4 \times 1 \times 1 \times 1$ grid partitioning has been utilized and therefore 4 skills need to be learned using LSB. After running LSB on the maze-world domain, it can be seen in Figure $5b$ that LSB significantly outperforms the monolithic approach. Note that each skill in LSB has the same parametric representation as the monolithic approach. That is, a five-tuple $\langle 1, x, y, \dot{x}, \dot{y} \rangle$. This simple parametric representation does not have the power to consistently solve maze-world using the monolithic approach. However, using LSB this simple representation is capable of solving the task in a near-optimal fashion as indicated on the average reward graph (Figure $5b$) and resulting value function (Figure $5c$).

We also tested LSB on the more challenging pinball-world domain ($5d$). The same LSB parameters were used as in maze-world, but the provided partitioning was a $4 \times 3 \times 1 \times 1$ grid. Therefore, 12 skills needed to be learned in this domain. More skills were utilized for this domain since the domain is significantly more complicated than maze-world and a more refined skill-set is required to solve the task. As can be seen in the average reward graph in Figure $5e$, LSB clearly outperforms the monolithic approach in this domain. It is less than optimal but still manages to sufficiently perform the task (see value function, Figure $5f$). The drop in performance is due to the complicated obstacle setup, the non-linear dynamics when colliding with obstacle edges and the partition design.

7 Discussion

In this paper, we introduced an iterative bootstrapping procedure for learning skills. This approach is similar to (and partly inspired by) skill chaining Konidaris and Barto [2009]. However, the heuristic approach applied by skill chaining may not produce a near-optimal policy even when the skill learning error is small. We provide theoretical results for LSB that directly relate the quality of the final policy to the skill learning error. LSB is the first algorithm that provides theoretical convergence guarantees whilst iteratively learning a set of skills in a continuous state space. In addition, the theoretical guarantees for LSB enable us to interlace skill learning with Policy Evaluation (PE). We can therefore perform PE whilst learning skills and still converge to a near-optimal solution.

In each of the experiments, LSB converges in very few iterations. This is because we perform policy evaluation in between each skill update, causing the global value function to converge at a fast pace. Initializing LSB in the partition class containing the goal state also results in value being propagated quickly to subsequent partition classes and therefore fast convergence. However, LSB can be initialized from any partition class.

One limitation of LSB is that it learns skills for all partition classes. This is a problem in high-dimensional state-spaces. However, the problem can be overcome, by focusing only on the most important regions of the state-space. One way to identify these regions is by observing an expert's demonstrations Abbeel and Ng [2005], Argall et al.

[2009]. In addition, we could apply self-organizing approaches to facilitate skill reuse Moerman [2009]. Skill reuse can be especially useful for *transfer learning*. Consider a multi-agent environment Garant et al. [2015] where many of the agents may be performing similar tasks which require a similar skill-set. In this environment, skill reuse can facilitate learning complex multi-agent policies (co-learning) with very few samples.

Given a task, LSB can learn and combine skills, based on a set of rules, to solve the task. This structure of learned skills and combination rules forms a *generative action grammar* Summers-Stay et al. [2012] which paves the way for building advanced skill structures that are capable of solving complex tasks in different environments and conditions.

One exciting extension of our work would be to incorporate skill interruption, similar to option interruption. Option interruption involves terminating an option based on an adaptive interruption rule Sutton et al. [1999]. Options are terminated when the value of continuing the current option is lower than the value of switching to a new option. This also implies that partition classes can overlap one another, as the option interruption rule ensures that the option with the best long term value is always being executed. Mankowitz et al. [2014] interlaced Sutton's interruption rule between iterations of value iteration and proved convergence to a global optimum. In addition, they take advantage of faster convergence rates due to temporal extension by adding a time-based regularization term resulting in a new option interruption rule. However, their results have not yet been extended to use with function approximation. Comanici and Precup [2010] have developed a policy gradient technique for learning the termination conditions of options. Their method involves augmentation of the state-space. However, the overall solution converges to a local optimum.

A Appendix

A.1 LSB Skill MDP

The formal definition of a Skill MDP is provided here for completeness.

Definition 5. *Given a target MDP $M = \langle S, A, P, R, \gamma\rangle$ and value function V_M, a* **Skill MDP** *for partition $\mathcal{P}_i$ is an MDP defined by $M'_i = \langle S', A, P', R', \gamma\rangle$ where $S' = \mathcal{P}_i \cup \{s_T\}$ where s_T is a terminal state and A is the action set from M. The transition probabilities*

$$P'(s'|s,a) = \begin{cases} P(s'|s,a) & \text{if } s \in \mathcal{P}_i \wedge s' \in \mathcal{P}_i \\ \sum_{y \in S \setminus \mathcal{P}_i} P(y|s,a) & \text{if } s \in \mathcal{P}_i \wedge s' = s_T \\ 1 & \text{if } s = s_T \wedge s' = s_T \\ 0 & \text{if } s = s_T \wedge s' \neq s_T \end{cases},$$

the reward function

$$R'(s,a) = \begin{cases} 0 & \text{if } s = s_T \\ \sum\limits_{s' \in \mathcal{P}_i} P(s'|s,a)R(s,a) & \text{if } s \neq s_T \wedge s' \neq s_T \\ \sum\limits_{y \in S \setminus \mathcal{P}_i} \psi(s,a,y) & \text{if } s \neq s_T \wedge s' = s_T \end{cases},$$

where $\psi(s,a,y) = P(y|s,a)\left(R(s,a) + \gamma V_M(y)\right)$, and γ is the discount factor from M.

A.2 Proof of Theorem 1

In this section, we prove Theorem 1.

We will make use of the following notations. For $m \geq 1$, we will denote by $[m]$ the set $\{1, 2, \ldots, m\}$. Let $\sigma = \langle \pi_\theta, \beta\rangle$ be a skill. Suppose the skill σ is initialized from a state s.

1. $P_\beta^{\pi_\theta}(s'|s,t)$ denotes the probability that the skill will terminate (i.e., return control to the agent) in state s' exactly $t \geq 1$ timesteps after being initialized.
2. $\widetilde{R}_{\beta,s}^{\pi_\theta}$ denotes the expected, discounted sum of rewards received during σ's execution. We use the $\widetilde{\cdot}$ notation to emphasize that this quantity is discounted.

The proof of Theorem 1 will make use of two lemmas. The first lemma (Lemma 1) demonstrates a relationship between the value of a skill policy before and after replacing a single skill. Within the skill's partition class there is a γ-contraction (plus some error), but outside the skill's partition class the value may become worse by a bounded amount. The second lemma (Lemma 2) uses Lemma 1 to prove that after a complete iteration (each skill has been update once), there is a γ-contraction (plus some error) over the entire state-space. We then prove Theorem 1 by applying the result of Lemma 2 recursively.

Lemma 1. *Let*

1. *M be the target MDP,*
2. *$\mathcal{P}$ a partition of the state-space,*
3. *μ be the skill policy defined by $\mathcal{P}$ (i.e., $\mu(s) = \arg\max_{i \in [m]} \mathbb{I}\{s \in \mathcal{P}_i\}$),*
4. *Σ be an ordered set of $m \geq 1$ skills, and*
5. *$i \in [m]$ be the index of the i^{th} skill in Σ.*

Suppose we apply $\mathcal{A}$ to the Skill MDP M'_i defined by M and $V_M^{\langle \mu, \Sigma\rangle}$, obtain π_θ, construct a new skill $\sigma'_i = \langle \pi_\theta, \beta_i\rangle$, and create a new skill set $\Sigma' = (\Sigma \setminus \{\sigma_i\}) \cup \{\sigma'_i\}$ by replacing the i^{th} skill with the new skill, then

$$\forall_{s \in S} \;\;,\;\; V_M^{\langle \mu, \Sigma\rangle}(s) - V_M^{\langle \mu, \Sigma'\rangle}(s) \leq \frac{\eta}{1-\gamma} \tag{5}$$

and

$$V^*_M(s) - V_M^{\langle\mu,\Sigma'\rangle}(s) \leq \begin{cases} \gamma \left\| V^*_M - V_M^{\langle\mu,\Sigma\rangle} \right\|_\infty + \frac{\eta}{1-\gamma} & \textit{if } s \in \mathcal{P}_i, \textit{and} \\ \left\| V^*_M - V_M^{\langle\mu,\Sigma\rangle} \right\|_\infty + \frac{\eta}{1-\gamma} & \textit{otherwise}, \end{cases} \tag{6}$$

where η is the skill learning error.

Proof.

Proving that (5) holds:

First, we show that (5) holds. For each skill $\sigma_i \in \Sigma$, we will denote the skill's policy and termination rule by π_i and β_i, respectively. If $s \in \mathcal{P}_j$ where $j \neq i$, then

$$\begin{aligned} V_M^{\langle\mu,\Sigma\rangle}(s) - V_M^{\langle\mu,\Sigma'\rangle}(s) &= \left(\widetilde{R}^{\pi_j}_{\beta_j,s} + \sum_{t=1}^{\infty} \gamma^t \sum_{s'} P^{\pi_j}_{\beta_j}(s'|s,t) V^{\langle\mu,\Sigma\rangle}(s') \right) - \left(\widetilde{R}^{\pi_j}_{\beta_j,s} + \sum_{t=1}^{\infty} \gamma^t \sum_{s'} P^{\pi_j}_{\beta_j}(s'|s,t) V^{\langle\mu,\Sigma'\rangle}(s') \right) \\ &\leq \gamma \left\| V_M^{\langle\mu,\Sigma\rangle} - V_M^{\langle\mu,\Sigma'\rangle} \right\|_\infty . \end{aligned}$$

On the other hand, if $s \in \mathcal{P}_i$, then

$$\begin{aligned} V_M^{\langle\mu,\Sigma\rangle}(s) - V_M^{\langle\mu,\Sigma'\rangle}(s) &= V_M^{\langle\mu,\Sigma\rangle}(s) + \left(V^*_{M'_i}(s) - V^*_{M'_i}(s) \right) - V_M^{\langle\mu,\Sigma'\rangle}(s) && \text{By inserting } 0 = \left(V^*_{M'_i}(s) - V^*_{M'_i}(s) \right) . \\ &= \left(V_M^{\langle\mu,\Sigma\rangle}(s) - V^*_{M'_i}(s) \right) + \left(V^*_{M'_i}(s) - V_M^{\langle\mu,\Sigma'\rangle}(s) \right) && \text{Regrouping terms.} \\ &\leq 0 + \left(V^*_{M'_i}(s) - V_M^{\langle\mu,\Sigma'\rangle}(s) \right) && \text{The definition of } M'_i \Rightarrow V^*_{M'_i}(s) \geq V_M^{\langle\mu,\Sigma\rangle}(s) . \\ &\leq \eta && \text{By Definition 4.} \end{aligned}$$

In either case,

$$V_M^{\langle\mu,\Sigma\rangle}(s) - V_M^{\langle\mu,\Sigma'\rangle}(s) \leq \gamma \left\| V_M^{\langle\mu,\Sigma\rangle} - V_M^{\langle\mu,\Sigma'\rangle} \right\|_\infty + \eta ,$$

which leads to (5) by recursing on this inequality.

Proving that (6) holds:

If $s \notin \mathcal{P}_i$, then by (5), we have

$$\begin{aligned} V^*_M(s) - V_M^{\langle\mu,\Sigma'\rangle}(s) &\leq V^*_M(s) - \left(V_M^{\langle\mu,\Sigma\rangle}(s) - \frac{\eta}{1-\gamma} \right) \\ &\leq \left\| V^*_M - V_M^{\langle\mu,\Sigma\rangle} \right\|_\infty + \frac{\eta}{1-\gamma} . \end{aligned}$$

Now we consider the case where $s \in \mathcal{P}_i$. Let $\sigma'_i = \langle \pi_\theta, \beta_i \rangle$ be the newly introduced skill. We will denote by $\sigma'_i\langle\mu,\Sigma'\rangle$ the policy that first executes σ'_i from a state $s \in \mathcal{P}_i$ and then follows the policy $\langle\mu,\Sigma\rangle$ thereafter.

$$\begin{aligned} \forall_{s \in \mathcal{P}_i} , \quad V^*_M(s) - V_M^{\langle\mu,\Sigma'\rangle}(s) &= V^*_M(s) - V_M^{\sigma'_i\langle\mu,\Sigma'\rangle} \\ &\leq V^*_M(s) - \left(V_M^{\sigma^*_i\langle\mu,\Sigma'\rangle} - \eta \right) \\ &= \left(\widetilde{R}^{\pi^*}_{\beta_i} + \sum_{t=1}^{\infty} \gamma^t \sum_{s'} P^{\pi^*}_{\beta_i}(s'|s,t) V^*_M(s') \right) \\ &\quad - \left(\widetilde{R}^{\pi^*}_{\beta_i} + \sum_{t=1}^{\infty} \gamma^t \sum_{s'} P^{\pi^*}_{\beta_i}(s'|s,t) V_M^{\langle\mu,\Sigma'\rangle}(s') \right) + \eta \\ &= \sum_{t=1}^{\infty} \gamma^t \sum_{s'} P^{\pi^*}_{\beta_i}(s'|s,t) \left(V^*_M(s') - V_M^{\langle\mu,\Sigma'\rangle}(s') \right) + \eta \\ &\leq \sum_{t=1}^{\infty} \gamma^t \sum_{s'} P^{\pi^*}_{\beta_i}(s'|s,t) \left(V^*_M(s') - \left(V_M^{\langle\mu,\Sigma\rangle}(s') - \frac{\eta}{1-\gamma} \right) \right) + \eta \\ &\leq \gamma \left\| V^*_M - V_M^{\langle\mu,\Sigma\rangle} \right\|_\infty + \gamma \left(\frac{\eta}{1-\gamma} \right) + \eta \\ &\leq \gamma \left\| V^*_M - V_M^{\langle\mu,\Sigma\rangle} \right\|_\infty + \frac{\eta}{1-\gamma} . \end{aligned}$$

□

Lemma 2. *Suppose we execute LSB for a single iteration. Let Σ be the set of skills at the beginning of the iteration and Σ' be the set of skills after each skill has been updated and the iteration has completed, then*

$$\left\| V_M^* - V_M^{\langle \mu, \Sigma' \rangle} \right\|_\infty \leq \gamma \left\| V_M^* - V_M^{\langle \mu, \Sigma \rangle} \right\|_\infty + \frac{m\eta}{1-\gamma} \quad . \tag{7}$$

Proof. Without loss of generality, we assume that the skills are updated in order of increasing index. We denote the skill set at the beginning of the iteration by Σ and the skill set at the end of the iteration by Σ' after all of the skills have been updated once. It will be convenient to refer to the intermediate skill sets that are created during an iteration. Therefore, we denote by $\Sigma_1', \Sigma_2', \ldots, \Sigma_m' = \Sigma'$, the set of skills after the first skill was replaced, the second skill was replaced, ..., and after the m^{th} skill was replaced, respectively.

We will proceed by induction on the skill updates. As the base case, notice that by Lemma 1, we have that

$$V_M^*(s) - V_M^{\langle \mu, \Sigma_1' \rangle}(s) \leq \begin{cases} \gamma \left\| V_M^* - V_M^{\langle \mu, \Sigma \rangle} \right\|_\infty + \frac{\eta}{1-\gamma} & \text{if } s \in \mathcal{P}_1, \text{and} \\ \left\| V_M^* - V_M^{\langle \mu, \Sigma \rangle} \right\|_\infty + \frac{\eta}{1-\gamma} & \text{otherwise.} \end{cases}$$

Let $1 \leq i < m$. Now suppose for Σ_i', we have that

$$V_M^*(s) - V_M^{\langle \mu, \Sigma_i' \rangle}(s) \leq \begin{cases} \gamma \left\| V_M^* - V_M^{\langle \mu, \Sigma \rangle} \right\|_\infty + \frac{i\eta}{1-\gamma} & \text{if } s \in \bigcup\limits_{j \in [i]} \mathcal{P}_j, \text{and} \\ \left\| V_M^* - V_M^{\langle \mu, \Sigma \rangle} \right\|_\infty + \frac{i\eta}{1-\gamma} & \text{otherwise.} \end{cases}$$

Now for Σ_{i+1}', we have several cases:

1. $s \in \mathcal{P}_{i+1}$:

   By applying Lemma 2, we see that

$$\begin{aligned} V_M^*(s) - V_M^{\langle \mu, \Sigma_{i+1}' \rangle}(s) &\leq \gamma \left\| V_M^* - V_M^{\langle \mu, \Sigma_i' \rangle} \right\|_\infty + \tfrac{\eta}{1-\gamma} \\ &\leq \gamma \left(\left\| V_M^* - V_M^{\langle \mu, \Sigma \rangle} \right\|_\infty + \tfrac{i\eta}{1-\gamma} \right) + \tfrac{\eta}{1-\gamma} \\ &\leq \gamma \left\| V_M^* - V_M^{\langle \mu, \Sigma \rangle} \right\|_\infty + \tfrac{(i+1)\eta}{1-\gamma} \quad . \end{aligned}$$

2. $s \in \bigcup\limits_{j \in [i]} \mathcal{P}_j$:

   By applying Lemma 2, we see that

$$\begin{aligned} V_M^*(s) - V_M^{\langle \mu, \Sigma_{i+1}' \rangle}(s) &\leq V_M^*(s) - V_M^{\langle \mu, \Sigma_i' \rangle}(s) + V_M^{\langle \mu, \Sigma_i' \rangle}(s) - V_M^{\langle \mu, \Sigma_{i+1}' \rangle}(s) \\ &\leq V_M^*(s) - V_M^{\langle \mu, \Sigma_i' \rangle}(s) + \tfrac{\eta}{1-\gamma} \\ &\leq \left(\gamma \left\| V_M^* - V_M^{\langle \mu, \Sigma \rangle} \right\|_\infty + \tfrac{i\eta}{1-\gamma} \right) + \tfrac{\eta}{1-\gamma} \\ &= \gamma \left\| V_M^* - V_M^{\langle \mu, \Sigma \rangle} \right\|_\infty + \tfrac{(i+1)\eta}{1-\gamma} \quad . \end{aligned}$$

3. $s \notin \bigcup\limits_{j \in [i+1]} \mathcal{P}_j$:

   Again, by Lemma 2, we see that

$$\begin{aligned} V_M^*(s) - V_M^{\langle \mu, \Sigma_{i+1}' \rangle}(s) &\leq \left\| V_M^* - V_M^{\langle \mu, \Sigma_i' \rangle} \right\|_\infty + \tfrac{\eta}{1-\gamma} \\ &\leq \left(\left\| V_M^* - V_M^{\langle \mu, \Sigma \rangle} \right\|_\infty + \tfrac{i\eta}{1-\gamma} \right) + \tfrac{\eta}{1-\gamma} \\ &\leq \left\| V_M^* - V_M^{\langle \mu, \Sigma \rangle} \right\|_\infty + \tfrac{(i+1)\eta}{1-\gamma} \quad . \end{aligned}$$

Thus by the principle of mathematical induction the statement is true for $i = 1, 2, \ldots, m-1$. After performing m updates, $\bigcup\limits_{j \in [m]} \mathcal{P}_j \equiv S$. Thus, we obtain the γ-contraction over the entire state-space.

□

A.2.1 Proof of Theorem 1

Proof. **(of Theorem 1)**

The loss of all policies is bounded by $\frac{1}{1-\gamma}$. Therefore, by applying Lemma 2 and recursing on (7) for $K \geq \log_\gamma(\varepsilon(1-\gamma))$ iterations, we obtain

$$\begin{aligned}
\|V_M^* - V_M^{\varphi}\|_\infty &\leq \gamma^K \left(\tfrac{1}{1-\gamma}\right) + \tfrac{m\eta}{(1-\gamma)^2} \\
&\leq \gamma^{\log_\gamma(\varepsilon(1-\gamma))} \left(\tfrac{1}{1-\gamma}\right) + \tfrac{m\eta}{(1-\gamma)^2} \\
&= (\varepsilon(1-\gamma)) \left(\tfrac{1}{1-\gamma}\right) + \tfrac{m\eta}{(1-\gamma)^2} \\
&= \tfrac{m\eta}{(1-\gamma)^2} + \varepsilon \ .
\end{aligned}$$

□

A.3 Experiments

We performed experiments on three well-known RL benchmarks: Mountain Car (MC), Puddle World (PW) Sutton [1996] and the Pinball domain Konidaris and Barto [2009]. The MC domain is discussed here. The PW and Pinball domains are found in the main paper. The purpose of our experiments is to show that LSB can solve a complicated task with a simple policy representation by combining bootstrapped skills. These domains are simple enough that we can still solve them using richer representations. This allows us to compare LSB to a policy that is very close to optimal. Our experiments demonstrate potential to scale up to higher dimensional domains by combining skills over simple representations.

Recall that LSB is a meta-algorithm. We must provide an algorithm for Policy Evaluation (PE) and skill learning. In our experiments, for the MC domain, we used SMDP-LSTD Sorg and Singh [2010] for PE and a modified version of Regular-Gradient Actor-Critic Bhatnagar et al. [2009] for skill learning.

In our experiments, for the MC domain, each skill is simply represented as a probability distribution over actions (independent of the state). We compare the performance to a policy using the same representation that has been derived using the monolithic approach. Each experiment is run for 10 independent trials. A 2×2 grid partitioning is used for the skill partition in this domain, unless otherwise stated. Binary-grid features are used to estimate the value function.

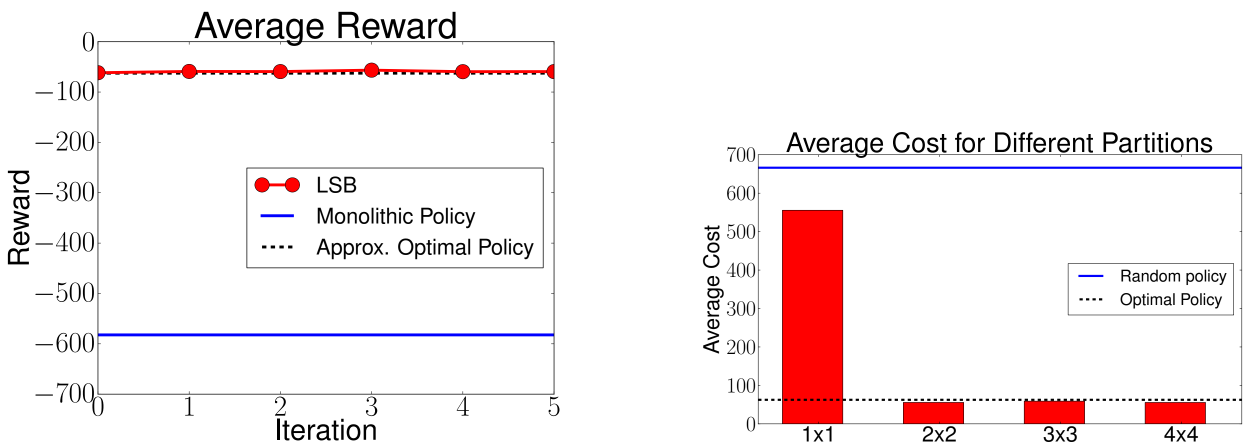
These are example representations. In principal, any value function and policy representation that is representative of the domain can be utilized.

A.3.1 Mountain Car

The Mountain Car domain consists of an under-powered car situated in a valley. The car has to leverage potential energy to propel itself up to the goal, which is the top of the rightmost hill. The state-space is the car's position and velocity $\langle p, v \rangle$.

Figure 6a compares the monolithic approach with LSB (for a 2×2 grid partition). The monolithic approach achieves low average reward. However, with the same restricted policy representation, LSB combines a set of skills, resulting in a richer solution space and a higher average reward as seen in Figure 6a. This is comparable to the approximately optimal average reward. Convergence is achieved after a single iteration since LSB is initiated from the partition containing the goal location causing value to be instantaneously propagated to subsequent skills.

Figure 6b compares the performance of different partitions where a 1×1 grid represents the monolithic approach. As seen in the figure, the cost is lower for all partitions greater than 1×1 which is consistent with the results in the main



(a) (b)

Figure 6: The Mountain Car domain: (a) The average reward for the LSB algorithm generated by the LSB skill policy. This is compared to the monolithic approach that attempts to solve the global task as well as an approximately optimal policy derived using Q-learning. (b) The average cost (negative reward) for different partitions (i.e., grid sizes).

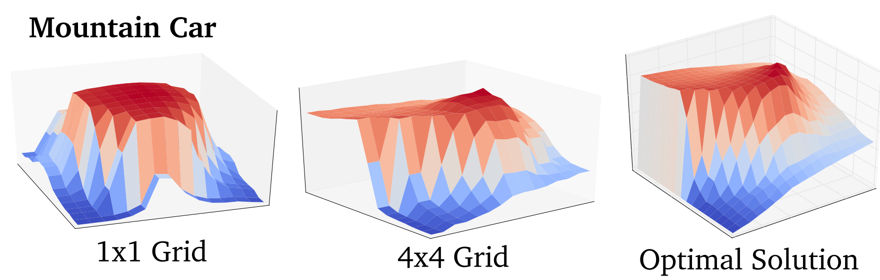


Figure 7: *The Mountain Car Domain*: Comparison of value functions for the monolithic approach (1×1 grid), the best partition using LSB (4×4 grid), and an approximately optimal value function (derived using Q-learning, applied for a huge number of iterations, with a fine discretization of each task's state-space).

paper. Figure 7 indicates the resulting value functions for various grid sizes. The value function from the monolithic approach is not capable of solving the task whereas the 4×4 grid partition's value function is near-optimal.

A.3.2 Puddle World

Figure 8 compares the value functions for different grid sizes in Puddle World. The monolithic approach (1×1 partition) provides a highly sub-optimal solution since, according to its value function, the agent must travel directly through the puddles to reach the goal location. The 3×3 grid provides a near-optimal solution.

A.4 Modified Regular-Gradient ActorCritic

We used a very simple policy gradient algorithm (Algorithm 2) for skill learning. The algorithm is based on Regular-Gradient ActorCritic Bhatnagar et al. [2009]. The algorithm differs from Regular-Gradient ActorCritic because it uses different representations for approximating the value function and the policy. For a state action pair $(s, a) \in S \times A$, a functions $\phi(s, a) \in \mathbb{R}^d$ and $\zeta(s, a) \in \mathbb{R}^{d'}$ mapped (s, a) to a vector with dimension d and d', respectively. We use the representation given by ϕ to approximate the value function and the representation given by ζ to represent the policy.

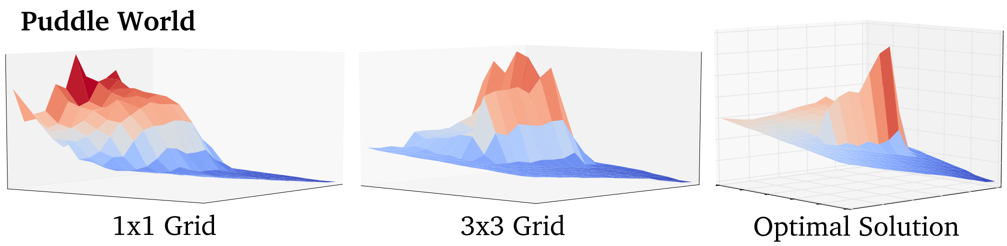


Figure 8: *The Puddle World Domain*: Comparison of value functions for the monolithic approach (1×1 grid), the best partition using LSB (3×3 grid), and an approximately optimal value function (derived using Q-learning, applied for a huge number of iterations, with a fine discretization of each task's state-space).

The parametrized policy was defined by

$$\pi_\theta(a|s) = \frac{\exp\left(\theta^T \zeta(s,a)\right)}{\sum_{a' \in A} \exp\left(\theta^T \zeta(s,a')\right)} \;, \tag{8}$$

where $\theta \in \mathbb{R}^{d'}$ are the learned policy parameters. We used representations such that $d' \ll d$, meaning that the policy parametrization was much simpler than the representation used to approximate the value function. This allowed us to get an accurate representation of the value function, but restrict the policy space to very simple policies.

Algorithm 2 Modified Regular-Gradient ActorCritic

Require:

1. ϕ : mapping from states to a vector representation used to approximate the value,
2. ω : value function approximation parameters,
3. ζ : mapping from states to a vector representation used to approximate the policy,
4. θ : policy parameters,
5. α : the value learning rate (fast learning rate),
6. β : the policy learning rate (slow learning rate, i.e., $\beta < \alpha$), and
7. (s, a, s', r) : a state-action-next-state-reward tuple.

1: $\widehat{V}_{\text{NEW}} \leftarrow \left(r + \gamma \sum_{a' \in A} \pi_\theta(a'|s')\omega^T \phi(s, a')\right)$ {Estimate V^{π_θ} given the new sample.}
2: $\widehat{V}_{\text{OLD}} \leftarrow \omega^T \phi(s,a)$ {Use the current value function approximation to estimate the value.}
3: $\delta \leftarrow \left(\widehat{V}_{\text{NEW}} - \widehat{V}_{\text{OLD}}\right)$ {Compute the temporal difference error.}
4: $\omega' \leftarrow \omega + \alpha\delta$ { Update the value function weights using the fast learning rate α. }
5: $\psi_{s,a} \leftarrow \zeta(s,a) - \sum_{a' \in A} \pi_\theta(a', s)\zeta(s, a')$ {Compute the "compatible features" Bhatnagar et al. [2009]. }
6: $\theta' \leftarrow \theta + \beta\delta\psi_{s,a}$ { Update the policy parameters using the slow learning rate β. }
7: **Return** $\langle \omega', \theta' \rangle$ { Updated value function and policy parameters. }

Although using different representations for approximating the value function and the policy strictly violates the policy gradient theorem Sutton et al. [2000], it still tends to work well in practice.

In our experiments, we used a fast learning rate of $\alpha = 0.1$ and a slow learning rate of $\beta = 0.2\alpha$. Value function and policy parameters were initialized to zero vectors.

A.5 Pinball Demonstration Videos

There are two videos attached showing a demonstration of a policy learned by LSB for the Pinball domain Konidaris and Barto [2009]. Both of these domains are analyzed and discussed in the main paper. The first video shows a policy learned for *Maze-world* and the second video shows a policy learned for *Pinball-world*, one of the standard pinball benchmark domains. The objective of the agent (blue ball) is to circumnavigate the obstacles and reach the goal region

(red ball). A colored square is superimposed onto the active skill indicating the current skill or partition class being executed.

References

Pieter Abbeel and Andrew Y Ng. Exploration and apprenticeship learning in reinforcement learning. In *Proceedings of the 22nd International Conference on Machine Learning*, 2005.

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009. ISSN 0921-8890.

Andrew Barto, George Konidaris, and C.M. Vigorito. Behavioral hierarchy: Exploration and representation. In *Computational and Robotic Models of the Hierarchical Organization of Behavior*, pages 13–46. Springer, 2013.

Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

Shalabh Bhatnagar, Richard S Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural actor–critic algorithms. *Automatica*, 45(11):2471–2482, 2009.

Justin A Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2-3):233–246, 2002.

Emma Brunskill and Lihong Li. PAC-inspired option discovery in lifelong reinforcement learning. *JMLR*, 1:316–324, 2014.

Gheorghe Comanici and Doina Precup. Optimal policy switching algorithms for reinforcement learning. In *Proceedings of the 9^{th} AAMAS*, pages 709–714, 2010.

Daniel Garant, Bruno C. da Silva, Victor Lesser, and Chongjie Zhang. Accelerating Multi-agent Reinforcement Learning with Dynamic Co-learning. Technical report, 2015.

Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the 14th Conference on Uncertainty in AI*, pages 220–229, 1998.

Ruijie He, Emma Brunskill, and Nicholas Roy. Efficient planning under uncertainty with macro-actions. *Journal of Artificial Intelligence Research*, 40:523–570, 2011.

George Konidaris and Andrew G Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *NIPS 22*, pages 1015–1023, 2009.

Daniel J Mankowitz, Timothy A Mann, and Shie Mannor. Time regularized interrupting options. *ICML*, 2014.

Timothy A Mann and Shie Mannor. Scaling up approximate value iteration with options: Better policies with fewer iterations. In *Proceedings of the 31^{st} ICML*, 2014.

Amy McGovern and Andrew G Barto. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. In *Proceedings of the 18th ICML*, pages 361 – 368, 2001.

Wilco Moerman. *Hierarchical reinforcement learning: Assignment of behaviours to subpolicies by self-organization*. PhD thesis, Cognitive Artificial Intelligence, Utrecht University, 2009.

Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21: 682–691, 2008.

Doina Precup, Richard S Sutton, and Satinder Singh. Theoretical results on reinforcement learning with temporally abstract options. In *ECML-98*, pages 382–393. Springer, 1998.

Martin L Puterman. *Markov Decision Processes - Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

Jonathan Sorg and Satinder Singh. Linear options. In *Proceedings* 9^{th} *AAMAS*, pages 31–38, 2010.

Douglas Summers-Stay, Ching Lik Teo, Yezhou Yang, C Fermuller, and Yiannis Aloimonos. Using a minimal action grammar for activity understanding in the real world. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4104–4111. IEEE, 2012.

Richard Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pages 1038–1044, 1996.

Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *AI*, 112(1):181–211, August 1999.

Richard S Sutton, David McAllester, Satindar Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063, 2000.

Vladimir Naumovich Vapnik. *Statistical learning theory*, volume 2. Wiley New York, 1998.